DOCUMENT RESUME

ED 056 535 EM 009 400

TITLE	Project Solo; Newsletter Number Nineteen.
INSTITUTION	Pittsburgh Univ., Pa. Dept. of Computer Science.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	29 Oct 71
NOTE	26p.; See also ED 053 566
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	Computer Programs; *Computer Science Education; Digital Computers; Grade 9; *Programing; Programing Languages
IDENTIFIERS	*Project Solo

ABSTRACT

A module designed to teach ninth grade students how to write simple machine language programs is offered in this newsletter of the University of Pittsburgh's Project Solo. The first few pages of the newsletter present a rationale for having ninth grade students learn programming, and some programs which were written by such students are presented. The relationship between machine language and a high-level language such as BASIC is then discussed. A general overview of the principle components of a digital computer and their relationships is presented, together with a discussion of the basic operation and use of a computer. These principles are then applied to a program, MICROCOMPUTER, which simulates a small computer. Instruction and data words are discussed, various instructions of the accumulator are described, and the functions of the control unit are provided. A sample program including a flow chart and procedures for signing onto and using MICROCOMPUTER are also included. (SH)

PROJECT SOLO*
AN EXPERIMENT IN REGIONAL COMPUTING FOR SECONDARY SCHOOL SYSTEMS

University of Pittsburgh • Department of Computer Science • Pittsburgh, Pennsylvania 15213

Newsletter No. 19 October 29, 1971

Inside Computers

One of the theoretical objections to using computers in education is that students might become "computer bums", i.e. they could conceivably get all caught up with computer science to the detriment of their other studies. This argument has some validity, although it sounds vaguely reminiscent of John Holt's story about the teacher who told the students to stop playing with the turtle because it was time to start studying biology.

The basic approach of Project Solo has been to emphasize the use of computers as tools which support both teacher and student in their learning activity. It has been our experience, however, that the truly inquisitive mind always wants to know about the tool as well as its use. When the tool is something as complex as a computer, the answers to such questions cannot be given in a few words. To help fill this gap, the Pittsburgh Public School system is working with Project Solo in the development of a 9th grade Computer Science course. In addition to supplying answers to the questions students ask about this new technology, we feel that such a course will eventually contribute to our original objective by bringing these youngsters up to a level of sophistication in the use of the computer as a tool that matches the sophistication of modern high school curricula. (Those who doubt this necessity should look at any good 12th grade mathematics or science text published recently.) We also feel that our 9th grade course can open up new career possibilities, as well as contribute to the motivation of students in tackling all of their learning opportunities.

MICRO COMPUTER

One of the units prepared for this new course is enclosed (Module #0100). The program, which simulates a very small computer that can only be programmed in machine language, should run with about any version of BASIC currently available. Oh yes...the simulation program was written by 9th grader Rob Drelles for us last summer. It took him all of one afternoon (we kid you not). Higher level versions (which permit such fancy features as "core" dumps) took him a little longer.

MAXI COMPUTER

But should high school students learn to program in such things as real assembly language? It all depends. Herewith is an example of a project that was put together from problem definition to final implementation by Howard Seltman, grade 12. (Howard has had one previous year of programming experience achieved through his use of computers in chemistry in a Project Solo school.)

The problem Howard attacked was to provide a measure of what the educational psychologists call "latency" in student response to CAI queries. This means that he had to arrange for the computer to calculate the time it took for a student to answer a query, supplying that time to the student or to a teacher file. It was also to be possible to use the time elapsed as a signal to interrupt the student response phase of the CAI cycle and take over control. Anyone who knows something about time-sharing will appreciate the complexity of this problem. A simple example of how Howard's program works is on the following page.

*Supported in part by NSF grant GJ-1077

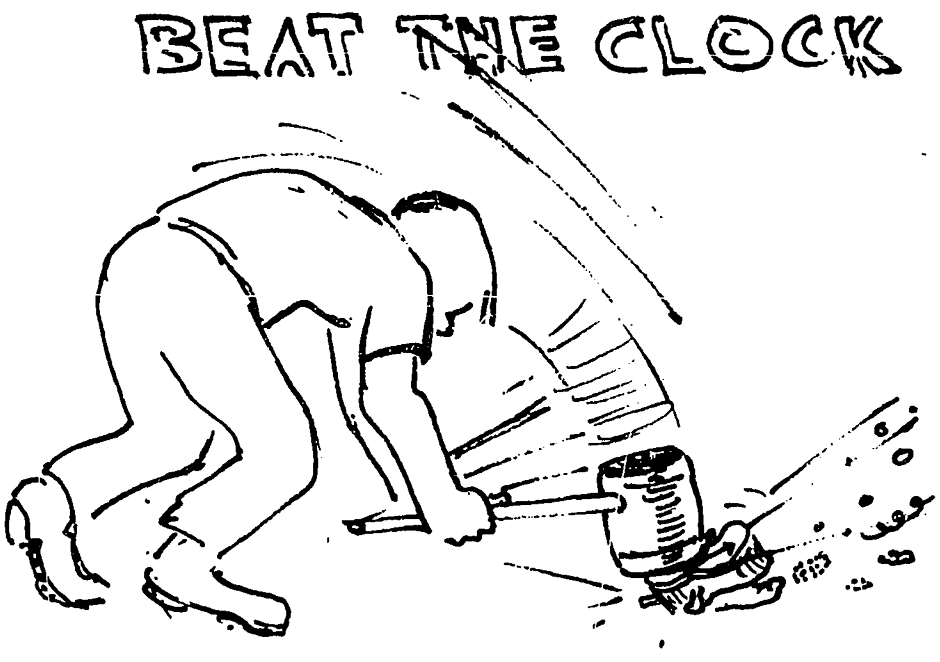
```
>LOAD /CAROLYN/
>RUN
SUBPROGRAMS REQUIRED
FILE:/TTIM/

ANSWER IN 5 SECONDS.
WHAT IS THE SQRT OF 100?10
YOU ANSWERED CORRECTLY IN 2 SECS.
WHAT IS THE SQRT OF 64?8
YOU ANSWERED CORRECTLY IN 4 SECS.
WHAT IS THE SQRT OF 81?9
YOU ANSWERED CORRECTLY IN 2 SECS.
WHAT IS THE SQRT OF 25?5
YOU ANSWERED CORRECTLY IN 2 SECS.
WHAT IS THE SQRT OF 121?
TOO SLOW, IT TOOK YOU MORE THAN 5 SECS.
WHAT IS THE SQRT OF 9?3
YOU ANSWERED CORRECTLY IN 4 SECS.
WHAT IS THE SQRT OF 4?2
YOU ANSWERED CORRECTLY IN 3 SECS.
WHAT IS THE SQRT OF 1?1
YOU ANSWERED CORRECTLY IN 1 SECS.
WHAT IS THE SQRT OF 49?
TOO SLOW, IT TOOK YOU MORE THAN 5 SECS.
WHAT IS THE SQRT OF 16?4
YOU ANSWERED CORRECTLY IN 2 SECS.
END OF DATA   LINE        40
```

The subprogram requested is the special assembly language program called TTIM which Howard wrote. Users on other numbers should request:

159WR /TTIM/

The subprogram capability illustrated here is one of the ways in which NEWBASIC can be extended for special users.



Listing of the NBS Program /CAROLYN/

```
10 INTEGER A,B,C,I
20 PR. "ANSWER IN 5 SECONDS."
40 READ I
41 A=5,B=0,C=0
50 PR. "WHAT IS THE SQRT OF":I:"?":
60 CALL TTIM(A,B,C)
70 IF A=-1 END
80 IF A=0 PR."TOO SLOW, IT TOOK YOU MORE THAN 5 SECS."GOTO 40
90 IF C=SQRT(I) PR. "YOU ANSWERED CORRECTLY IN":B:" SECS."GOTO 40
100 PR. "YOU ANSWERED INCORRECTLY IN ":B:"SECS."
110 PR. "THE ANSWER IS":SQRT(I):"."
120 GOTO 40
200 DATA 100,64,81,25,121,9,4,1,49,16
```

The subroutine /TTIM/ used here was written in XAP. XAP is a combination of XTRAN (extended FORTRAN) and the SDS 940 assembler, TAP. One of the advantages of using a full-blown system like Com-Share is the availability of other processors which can do more subtle things than BASIC or NEWBASIC. The beauty of NEWBASIC is that it can call on an "infinite" supply of auxiliary routines of this sort.

What's a Teacher to Do?

One last comment. The teacher who has gotten this far may have thrown up his or her hands in despair. How can computers possibly be used by hard working teachers who don't begin to have the time to keep up with something like the above? It strikes us that this is not a problem at all, but a rare opportunity to improve the educational process. The president of a big industrial firm does not worry about not knowing how to apply differential equations to engineering problems. Quite the contrary. He understands the importance of his role as a manager, and the value of his setting goals to guide a multitude of talents. And so it should be for the teacher. The tendency for computer technology to bring out this managerial role in educators can be a very big plus. We see more and more teachers exploiting this opportunity, with some very fine teacher-student relationships developing as a result.

```
SUBROUTINE TTIM(ISEC,ITIM,IRESULT)
INTEGER I(20)
ON ESCAPE:100
ON BELL:100
FOR K=1,20: I(K)=0
ENTER XAP
BRS 11
LDA =20000777B
BRS 55
BRS 14
BRS 42
STA ITIME
STA JTIME1
LDA* ISEC
MUL =60
RSH 1
COPY BA
ADM ITIME
COPY X
COPY A
STA INDEX
STA INUM
STA ICOUNT
STA ITEMP
.L2 BRS 13
BRU .L9
BRS 42
SKL ITIME
BRU .L6
BRU .L2
.L9 TCI ITEMP
SKU =147B
BRU .L14
SKU =141B
BRU .L5
SKU =167B
BRU .L26
SKU =161B
BRU .L26
SKU =155B
BRU .L3
MIN INDEX
LDX INDEX
STA I,2
BRU .L2
.L5 LDX INDEX
COPY A
STA I,2
LDA =-1
ADM INDEX
BRU .L2
.L26 COPY A
ENTER XTRAN
FOR K=1,INDEX: I(K)=0
ENTER XAP
COPY A
STA INDEX
BRU .L2
.L3 BRS 42
STA JTIME2
.L27 MIN ICOUNT
LDX ICOUNT
LDA I,2
SKG =17B
BRU .L19
SKL =32B
BRU .L19
LDA INUM
MUL =12B
RSH 1
COPY BA
ADD I,2
SUB =20B
STA INUM
BRU .L27
.L14 LDA =-1
BRU .L18
.L19 LDA INUM
STA* IRESULT
LDA JTIME2
SUB JTIME1
STA JTIME
ENTER XTRAN
JTIME=JTIME/60
ENTER XAP
LDA JTIME
STA* ITIM
LDA =1
BRU .L18
.L6 LDA* ISEC
STA* ITIM
COPY A
.L18 STA* ISEC
LDA =155B
SKE ITEMP
TCO =155B
ENTER XTRAN
100 C'E
ENTER XAP
LDA =40000600B
BRS 55
ENTER XTRAN
RETURN
END
```

EXPLANATION OF THE SUBROUTINE /TTIM/ (by Howard Seltman)

The first part of this program, to read elapsed time of an input statement, is relatively simple. The system clock is read via a system programmed operator. When the carriage return is recognized, the system clock is again read. The difference between the two times is divided by 60 (because the system clock is incremented every 1/60 of a second) and the elapsed time in seconds is arrived at. To make an interrupt possible after a specified number of seconds is more difficult. First, every character must be made a break character, again by a SYSPOP. The reason for this is that after a terminal character input statement (TCI) the computer waits for input until it recognizes a break character. The value of the system clock when the specified number of seconds is elapsed is then calculated. Next a loop is initiated which checks for a character in the input buffer and then checks if the time is up. If time is up, a 0 is stored in the first argument and the subroutine returns to the calling program. If a character is seen in the input buffer, a single character is read with a TCI. This can be done because all characters are break characters. TCI simply reads the internal ASCII value of the character. This must then be processed. If the character is a control-G, a -1 is stored in the first argument as an indicator and the subroutine returns. If an editing character (control A, W, or Q) is recognized a branch is made to editing routines. Editing must be done "by hand" because this is not a "normal" input statement. If the character is not a carriage return, it is stored in an array and the check-buffer check-time loop is restarted. If the character is a carriage return, it is converted from internal ASCII to the actual base 10 value of the inputed digits. A 1 is stored in the first argument to indicate successful input within the allowed time. The converted input is stored in the third argument and the subroutine returns. One special problem must also be dealt with: in no case can the subroutine be allowed to return without setting the terminal status back to normal, i. e. control characters again become the only break characters and editing is done by the system.

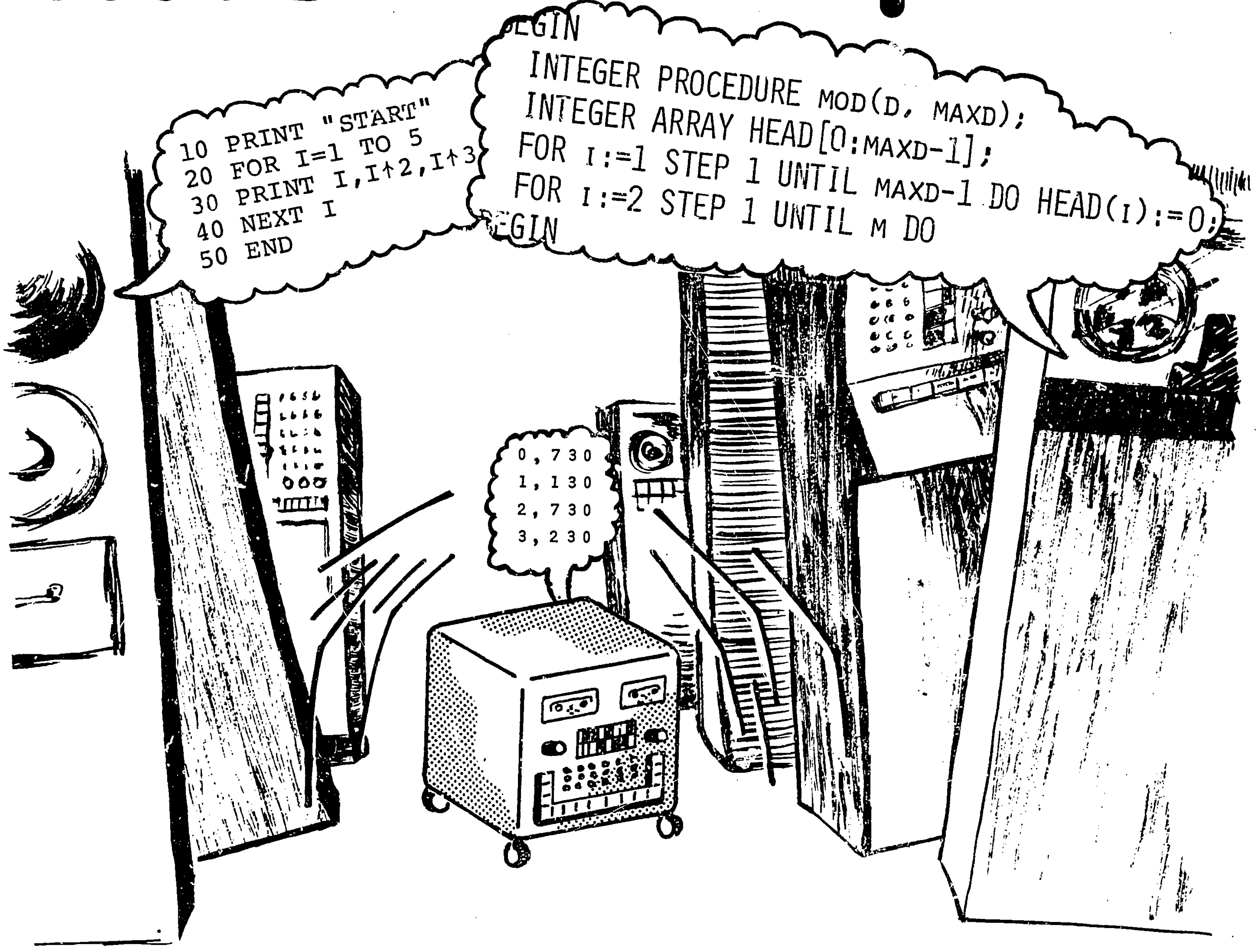
Newsletter Reminder:

Have you sent in your renewal form? (see Newsletter No. 18)

P.S.: Free loan of the film "Project Solo" is now being handled by Association-Sterling Films through their office located at 324 Delaware Avenue, Oakmont, Pa. 15139. Write them directly for a booking.

PROGRAMMING IN MACHINE LANGUAGE ON

MICRO.computer



```
10 PRINT "START"
20 FOR I=1 TO 5
30 PRINT I,I↑2,I↑3
40 NEXT I
50 END
```

```
BEGIN
  INTEGER PROCEDURE MOD(D, MAXD);
  INTEGER ARRAY HEAD[0:MAXD-1];
  FOR I:=1 STEP 1 UNTIL MAXD-1 DO HEAD(I):=0;
  FOR I:=2 STEP 1 UNTIL M DO
BEGIN
```

```
0, 730
1, 130
2, 730
3, 230
```

#0100
PROJECT SOLO
Computer Science Department
University of Pittsburgh 15213

● This module will help you understand what <u>really</u> goes on inside a computer when it is told to run a program written in a higher level language like BASIC or NEWBASIC.

● After giving you a general picture of the internal organization of a digital computer, the module will then allow you to write simple machine language programs, using a "simulated" machine called MICRO-COMPUTER. The prerequisite for this module is some experience with programming, and a lot of curiosity.

INSIDE COMPUTERS

Learning to Write Machine-Language Programs
for the MICRCOMPUTER

1. INTRODUCTION

Computers cannot actually "understand" high-level languages such as BASIC or FORTRAN. Because of their electronic nature they must have instructions given to them in a more primitive form which is called machine language.

Since the statements of machine languages are strings of numbers, writing programs in them is at best a tedious and complicated task. For this reason most people write their programs in a high-level language (like BASIC), and then let the computer itself translate their instructions into machine language. The translator is really another program (permanently stored in the computer) which is usually called a compiler.

In this module you will:

a) Study a general overview of the organization of a digital computer.

b) See how this applies to a specific example called the MICROCOMPUTER (actually this is not a real computer, but a program which simulates a very small computer).

c) Learn to write machine language programs for the MICROCOMPUTER. In this way you will see what life would be like without a high level language and compiler. You will also get a much better feel for what really goes on inside real computers.

Although the MICROCOMPUTER is a very small and limited device compared to real-life computers, it is possible to write interesting and powerful programs for it.

2. THE ORGANIZATION OF COMPUTERS

The principal components of a digital computer and their relationships are illustrated in Figure 1.

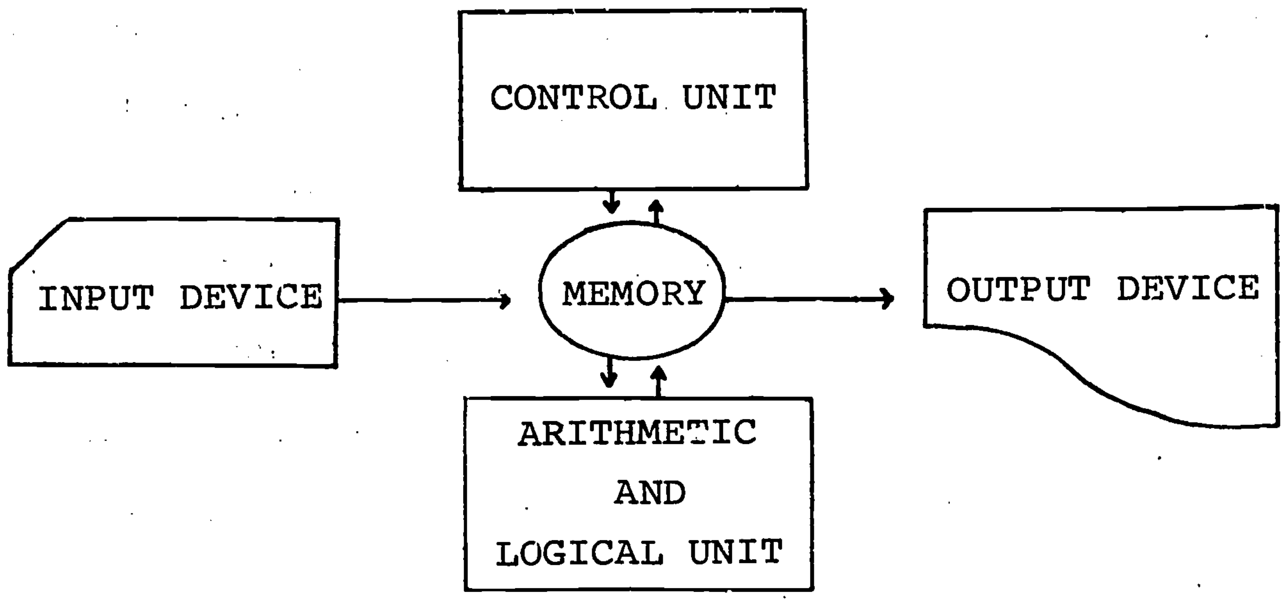


FIGURE 1.

The <u>Control Unit</u>--controls the sequence of events within the computer by interpreting and causing the execution of coded instructions received from the memory. (If this sounds vague, read on. It should become clearer.)

The <u>Arithmetic and Logical Unit</u>--performs the four arithmetic operations on data sent to it from the memory unit (+,-,*,/); it also controls "branching" by testing for negative and zero values.

The <u>Memory</u>--contains the instructions to be executed <u>and</u> the data on which the operations are to be performed.

The Input Device--accepts information, both instructions and data, and stores it in the memory. Examples of input devices include teletype keyboards, paper tape readers and punched card readers.

The Output Device--receives results from the memory and presents them to the user. Paper tape punches, teletype and line printers, and plotters can all serve as output devices.

3. BASIC OPERATION AND USE OF THE COMPUTER

The memory of a computer consists of a sequence of units called words, each of which can store the same amount of information. This amount of information varies from computer to computer but is typically something like a 10 decimal digit number. Each word is individually addressable; that is, each word can be referenced by a unique address. These addresses are numbers which start with zero and go up to one less than the number of words in the memory. It is very important to keep clear the difference between the address of a word, and the contents of a word. (See page 6)

There are two properties which are common to all computer memories, these are destructive read in and nondestructive read out. What the first term means is that each time information is stored in a given word in memory, the contents of that word are erased as the new information is "read in". Non-destructive read-out means that when information is "read out" from a given word of memory, the contents of the memory location itself are undisturbed. Hence, information may be "read out" of a given memory location as many times as needed, without destroying the contents of the word itself.

The key to the operation of the computer is the component referred to above as the "control unit". This device is capable of retrieving the contents of those words in memory that describe the "program" to be run. Each time such information is brought into the control unit it is interpreted as an instruction which is then executed.

The execution of an instruction involves the use of one or more of the other components of the computer. For example, an arithmetic instruction will require the use of the memory and the arithmetic and logical unit; an input instruction will involve an input device and the memory.

In any case, the use of a computer consists of a process of placing in memory a sequence of instructions in the order in which they will be fetched, interpreted and executed. A string of instructions so arranged that their execution accomplishes a particular task is called a program.

The instructions themselves are indistinguishable from any other kind of information in memory. They are recognized and interpreted as instructions only when they are brought into the control unit. Each type of computer is distinguished by its own repertoire or list of instructions which the control unit is capable of interpreting and executing. The repertoire is fixed and is determined by the electronic circuitry of the computer.

4. THE MICROCOMPUTER

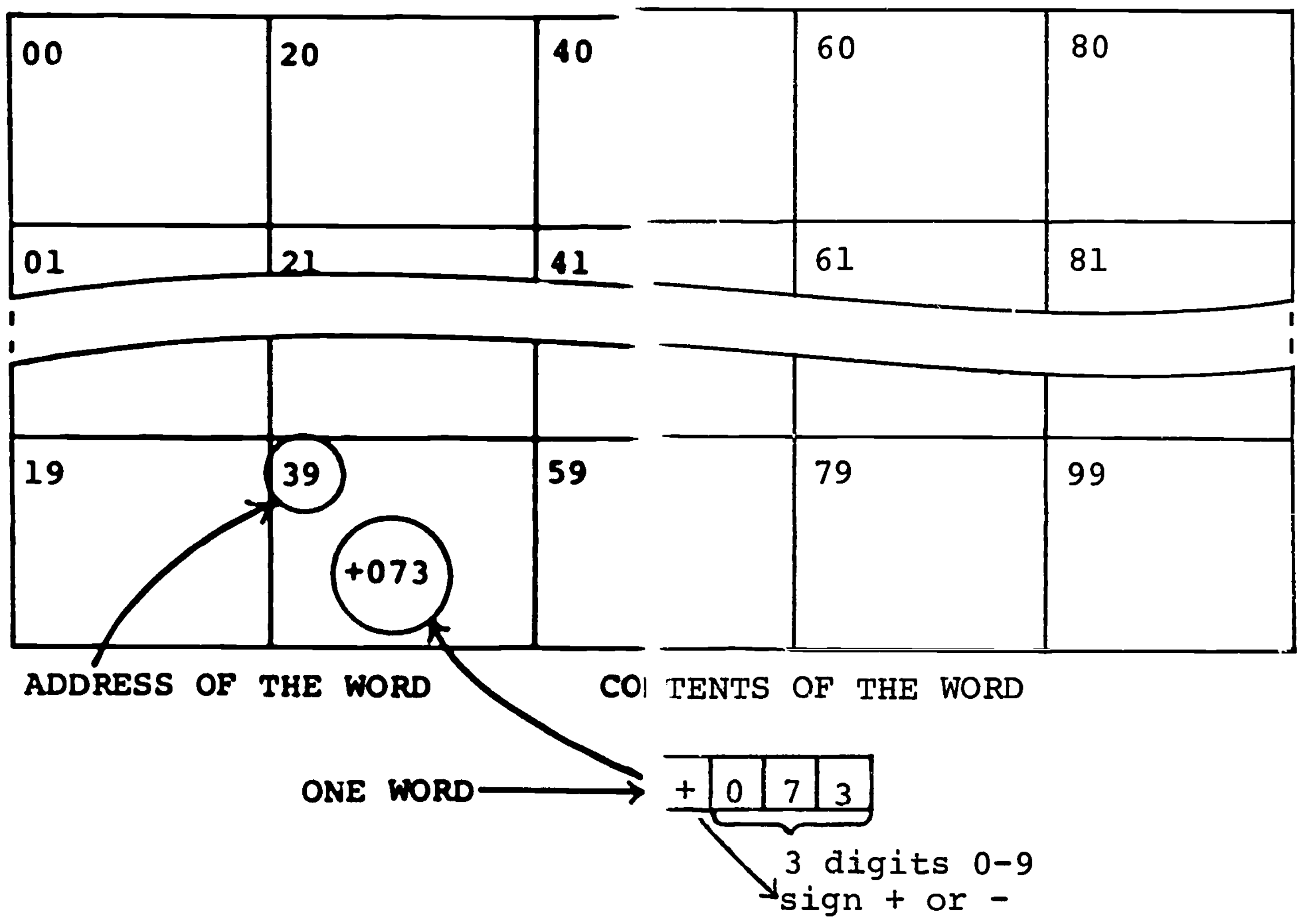
Let us now look at a specific case called the MICROCOMPUTER to see how the general principles discussed above apply to it. We will use decimal digits to represent the instructions and data.

In reality, the MICRCOMPUTER is not an actual "hardware" computer. It is a program written in NEWBASIC which simulates a small computer. You do not have to understand what simulation means at this point. Just tell yourself that when you run the program /MICRO/ you can think of the teletype as the input and output devices of a "make believe" computer whose design is described below:

The Memory

The memory consists of 100 words addressed (numbered) from 00 to 99. Each word will have the capacity to store three decimal digits and a sign as shown in Figure 2 on the following page.

FIGURE 2.



Although every word in ou MICROCOMPUTER is limited to storing a sign and 3 digit num er, the meaning of these digits for the computer is not always the same. Sometimes we want the computer to think of this umber as data, i.e., an actual number between -999 and +999 w ich will be used in a calculation.

At other times, we will w nt the computer to view the contents of a word as an instr ction. Each instruction will also consist of a three digit ode but the sign position of the word in which it is sto ed will be ignored.

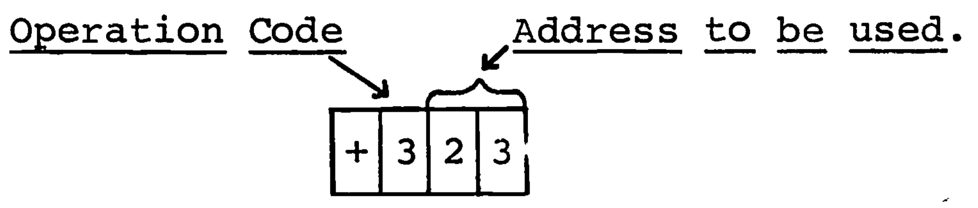
The Format of an Instruction Word

An instruction word will contain 1) a single digit operation code from 0 to 9 (there is one for each of the 10 instructions in the repertoire) and 2) the address of the word which is to be used in execution of the instruction. The way in which the MICROCOMPUTER knows which words in memory are instructions words will become clear in the discussion of the Control Unit. The instructions are summarized in Table 1 and described in detail in the next three sections.

TABLE 1

Instruction Repertoire for the MICROCOMPUTER

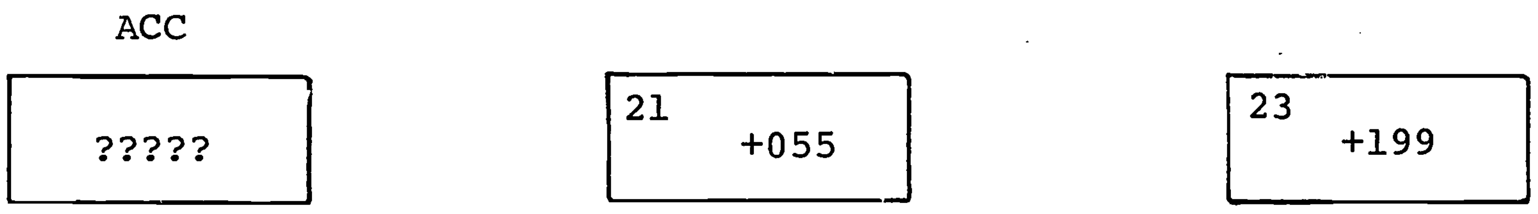
Operation Code	Action
1	Clear and add
2	Add
3	Subtract
4	Store Accumulator
5	Multiply
6	Divide
7	Read
8	Print
9	Transfer Unconditionally
0	Transfer on Minus



The Arithmetic Unit

The arithmetic unit consists of a device called the ac-cumulator whose purpose is to hold the results of individual additions, subtractions, multiplications, and divisions. The accumulator has the capacity to hold a sign and four digits. As an example of how the accumulator works consider the task of adding the two numbers found in words 21 and 23 of memory. Assume that the values found in these locations are +055 and +199.

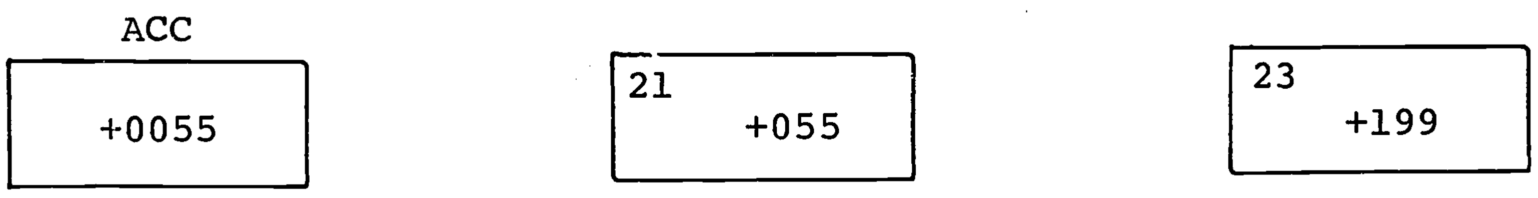
Hence, the situation can be represented as:



After executing the instruction

1 21

we have



The clear and add instruction, whose operation code is 1 makes the accumulator zero then adds to it the contents of the designated word of memory. Note that the contents of word 21 are unchanged (non-destructive read out).

Summary: When you give the MICROCOMPUTER the command 121, what you are telling it is "change the contents of the accumulator to zero, then add to the accumulator a "copy" of the number found in memory location 21".

Now let's execute

2 23

After which we have

ACC	21	23
+0254	+055	+199

The <u>add</u> <u>instruction</u>, whose operation code is 2, simply adds to the accumulator the contents of the specified word and places the result in the accumulator. If the absolute value of the sum is greater than 9999, the accumulator is set to zero, an overflow warning message is typed out, and the program continues running.

Subtraction is accomplished with an instruction whose operation code is 3. When a <u>subtract</u> <u>instruction</u> is executed, the contents of the specified word are subtracted from the accumulator. Again the result is placed in the accumulator and an overflow message is printed if necessary.

The operation code for <u>multiplication</u> is 5. The contents of the memory word and the contents of the accumulator are mutliplied, the result is stored in the accumulator, and an overflow warning is written if the result is too large.

Division is slightly different because of the fact that fractions cannot be represented within a single word of the MICROCOMPUTER. When a _divide instruction_--one with an operation code of 6 is executed, the MICROCOMPUTER takes the contents of the accumulator as the dividend, and the contents of the specified memory word as the divisor. The quotient is placed in the accumulator but the remainder is lost (ignored).

The _store accumulator instruction_ (operation code 4) provides the means to return the results of the arithmetic operations to the memory. This allows you to save a number for future use, while using the accumulator for other purposes. Notice however that the accumulator can contain 4 digit signed numbers but the memory words can only contain 3 digit signed numbers. If an attempt is made to store a number with absolute value greater than 999 an error error message will be typed and the program stopped.

The Control Unit

In the above discussion of the arithmetic unit it was not clear where the instructions that were executed came from or how the order of their execution was determined. This is the function of the control unit. In the MICROCOMPUTER, the control unit has two components. (1) The instruction register (IR) has a three digit capacity to hold the instruction that is being examined and executed. (2) The instruction counter (IC) has a two digit capacity to hold the address of the next instruction to be brought from memory. When the action of one instruction is completed, the control unit will bring the next instruction from the memory address given by the IC and place it in the IR. While this is going on, the number in the IC is increased by 1.

Figure 3 illustrates this sequence of events for a stored program which executes the same instruction sequence discussed above in the section on the arithmetic unit (pages 8 and 9).

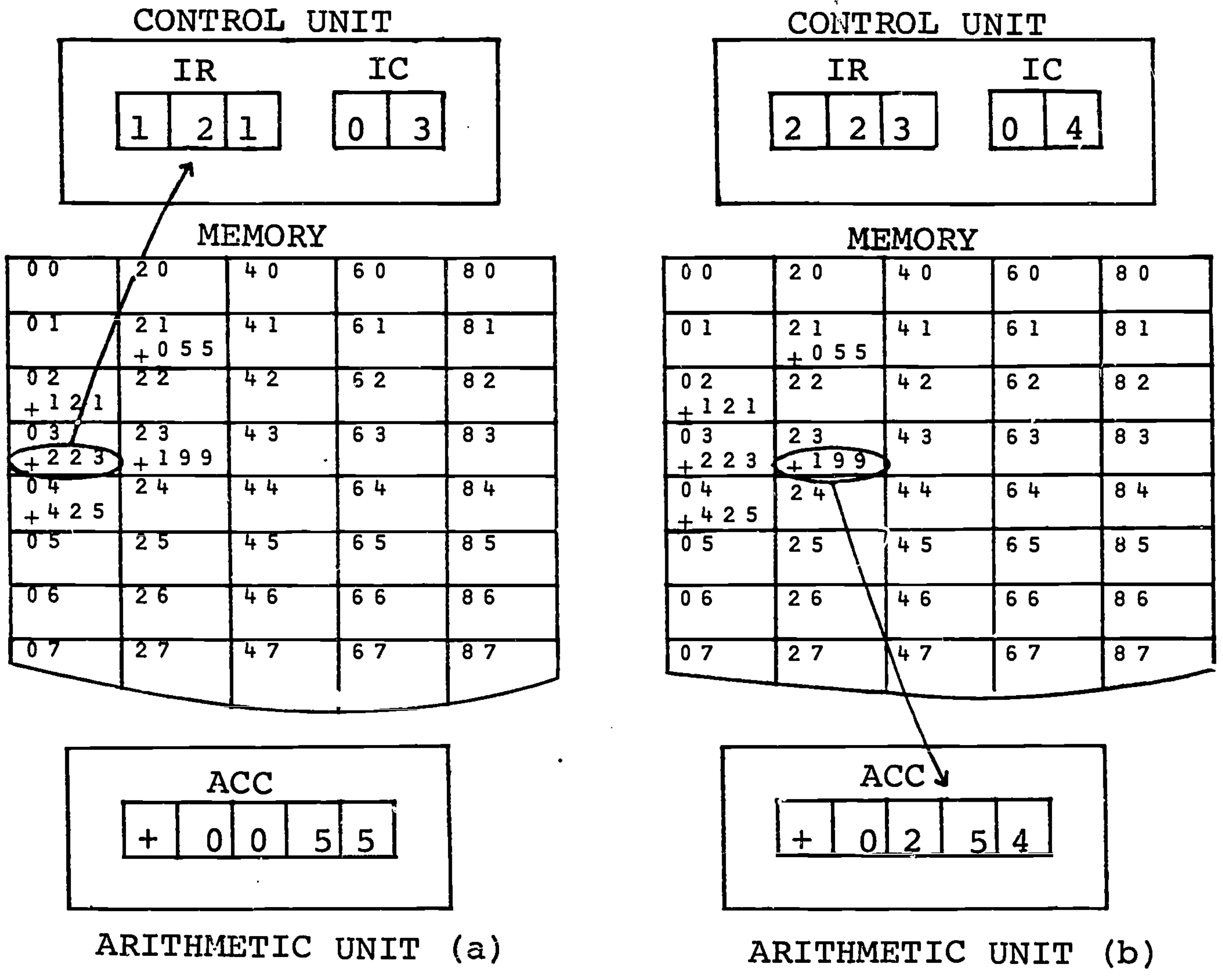


FIGURE 3.
a) The instruction in the IR has just been executed leaving +0055 in the ACC. The instruction of 03 is being brought from memory to be placed in the IR (destructive read in). While this is going on, the number in the IC will be increased by 1. b) The instruction from 03 is examined and executed, resulting in a new value, +0254 in the ACC.

NOTE: The above program starts in the location with address 02. Actually, we could have "loaded" the program anywhere we wished in memory.

Two very useful instructions which are closely related to the IC are the Transfer Unconditionally Instruction (Operation Code 9) and the Transfer on Minus Instruction (Operation Code 0). A computer's power lies largely in its ability to execute sequences of instructions over and over, or to choose between two possible instructions depending on the result of some previous operation. The two transfer instructions make these things possible in the MICROCOMPUTER. Transfer Unconditionally causes the IC to receive the address which is specified in the instruction. Hence, the next instruction to be executed after the Transfer Unconditionally is the instruction located in the word whose address was specified in the Transfer Unconditionally instruction. Figure 4 may help clarify this.

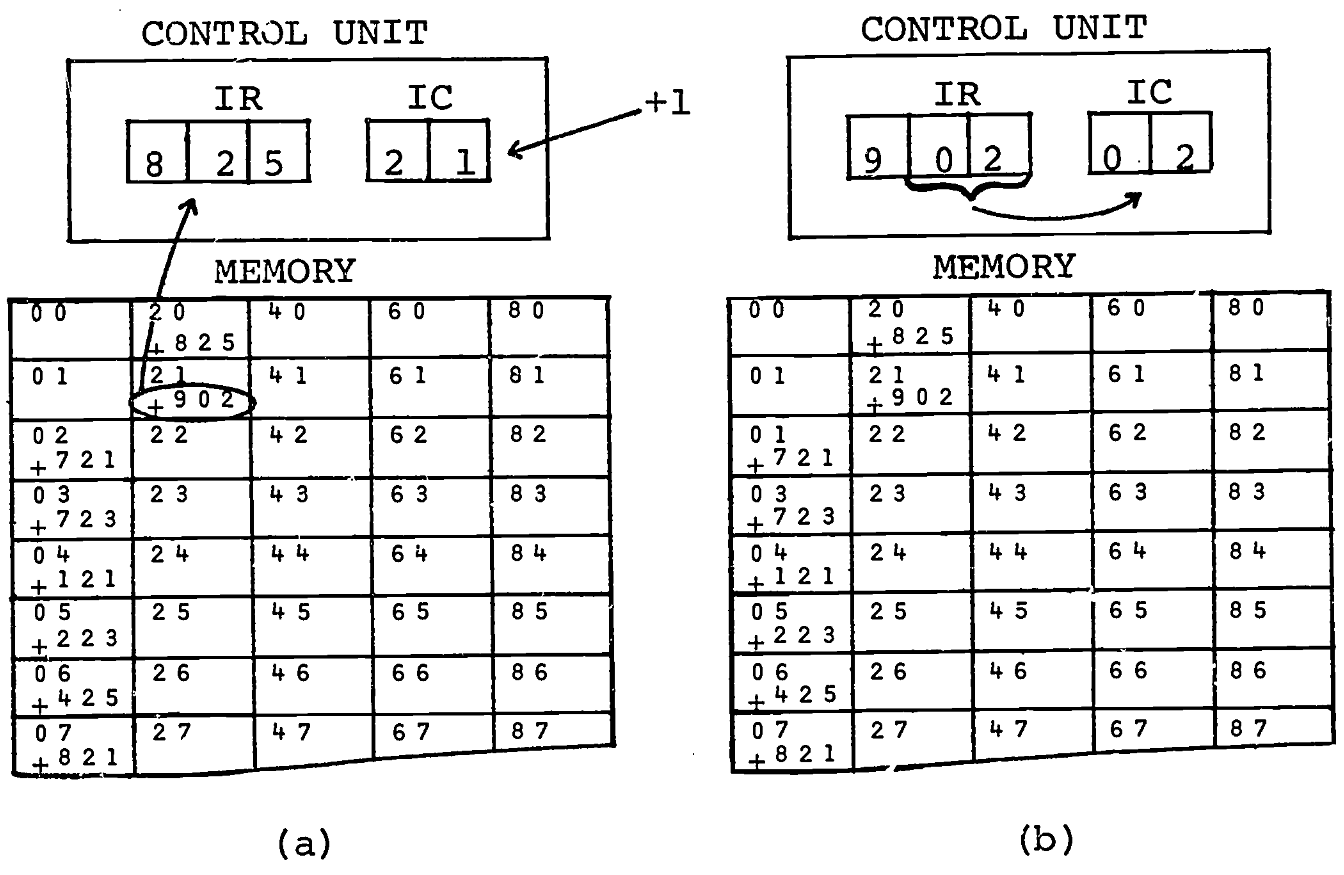


FIGURE 4.

Figure 4. The IC initially contains 21 so the instruction in word 21, namely 902, is brought into the IR (Figure 4a). But 9 is the operation code for Transfer Unconditionally instruction hence when the instruction is executed, the IC gets the address part of the instruction--that is 02 (Figure 4b). Then the next instruction to be loaded into the IR will be 721 in word 02, and the program will continue from there.

The Transfer on Minus instruction is similar, but the IC receives the address of the instruction only if the accumulator contains a negative number. If the accumulator is not negative, the IC has its content increased by 1 in the normal manner.

Input/Output

A computer is useless unless the results of its work can be printed out and unless it can receive data from the outside world. An instruction is provided in the MICRO-COMPUTER's repertoire called the Read Instruction with an operation code of 7 which causes a number typed into the teletype keyboard to be stored in the memory location specified in the instruction. For example, suppose the IC contains 05 and word 05 of the memory contains 723. After the instruction is executed the IC will contain 6, word 05 will be the same, and word 23 will contain whatever number was typed on the teletype.

Suppose now that word 06 contains 823. The operation code 8 means print. The same number that was read will be printed back onto the teletype paper.

A Sample Program

The flow chart in Figure 5 represents a procedure for reading two numbers, computing the absolute value of their difference, and printing the result. A possible program to accomplish the procedure is listed as it would appear in the MICROCOMPUTER in Table 2. You should follow this program step by step to gain an added grasp of what goes on in the computer by doing the following:

1. Draw the memory, arithmetic unit, and control unit as in Figures 2,3, and 4.
2. Enter the instructions in their proper squares of the memory.
3. Place 09 in the IC. (09 was arbitrarily chosen as the location of the first instruction).
4. Execute the instruction in word 09, update the IC, execute the next instruction, etc.

Remember that when a transfer instruction is executed, the IC may be loaded with the address of some instruction rather than being increased by 1.

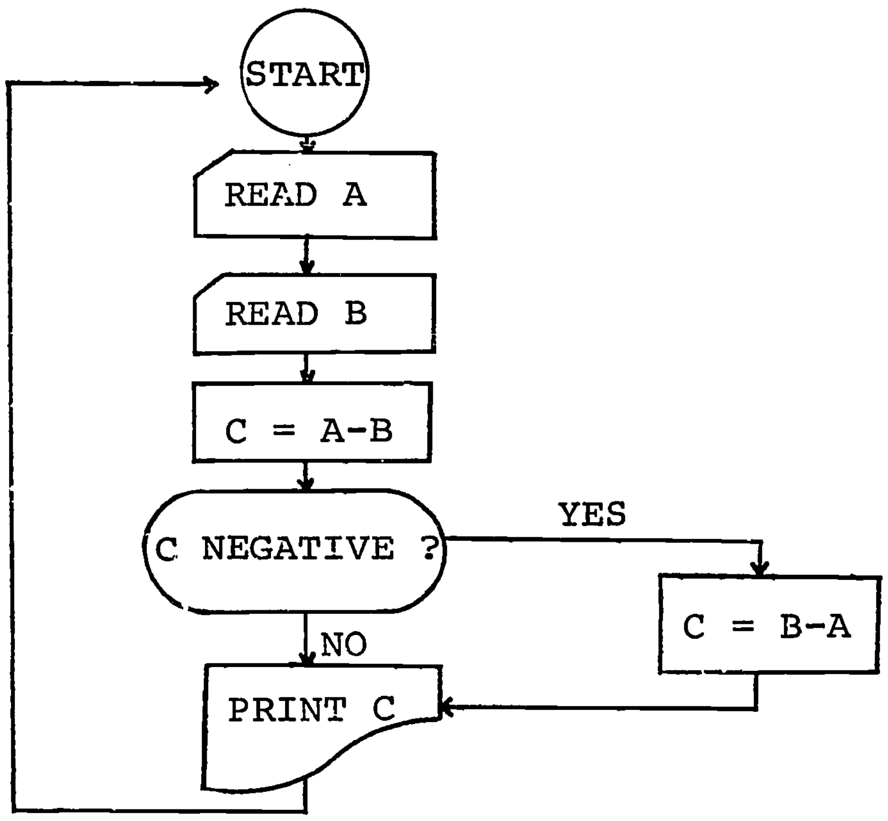


FIGURE 5.

TABLE 2

Program for Computing $|A-B|$ (See note 2 below)

Address	Instruction	Explanation
09	701	Read A
10	702	Read B
11	101	Store A in the Accumulator
12	302	Subtract B
13	403	Store C=A-B in word 03
14	017	Transfer on Minus to 17
15	803	Print C
16	909	Transfer to 09 (start again)
17	102	Store B in the Accumulator
18	301	Subtract A
19	403	C=B-A in word 03
20	915	Transfer to 15 (Print)

Note 1: Instructions begin at word 09. A is stored at 01, B at 02, C at 03.

Note 2: The mathematical notation $|A-B|$ is sometimes also written as "Absolute value of A-B" or ABS(A-B).

ABS(A-B) is defined as the number you get by substracting the smaller of the two numbers A and B from the larger of the two numbers.

Examples: ABS(5-3) = $|5-3|$ = 5-3 = 2
ABS(3-5) = $|3-5|$ = 5-3 = 2

5. OPERATION OF THE MICROCOMPUTER

To run programs on the MICROCOMPUTER you must first log in and enter NBS (recall that the MICROCOMPUTER is simulated by a NEWBASIC program). Then type >RUN 106UØP /MICRØ/.

The MICROCOMPUTER will print a header message and an asterisk (*). When this happens you can do any of the following:

a) Enter or change a memory word by typing the address, a comma, the contents of the word, and return. For example, *16,201(CR) would put +201 in word 16.

b) Begin execution of a program in the memory by typing 100, a comma, the address of the first word to be executed, and return. This sets the IC and starts the MICROCOMPUTER.

c) Examine the contents of a word by typing 101, comma, the address of the word, and return.

d) Leave the system (turn off the MICROCOMPUTER) by typing 102, comma, 0, return.

On the following page is a sample run of the program discussed in the previous section. Because of the way in which the program was written the only way to make it stop is to type ESC. Look at the flowchart to see why.

This is the program to calculate ABS(A-B). Your program for A Sample Program should closely resemble the sample run below:

```
PLEASE LOG IN:I113;PIT-------------------------LOG IN
READY, SYSTEM W04
 SEP 29  9:00
LAST LOGIN  SEP 29  8:58
-NBS-------------------------------------------ENTER NBS
VER. AUG 26  9:26
>RUN 113PIT /MICRO/----------------------------RUN THE MICROCOMPUTER

MICROCOMPUTER VERSION 5 LEVEL 1 SEP 29 09:01
*9,701  --|
*10,702   |
*11,101   |
*12,302   |
*13,403   |
*14,017   |
*15,803   |------------ ENTER INSTRUCTIONS AND/OR DATA INTO THE MEMORY
*16,909   |
*17,102   |
*18,301   |
*19,403   |
*20,915 --|
*100,9----------------- SET THE INSTRUCTION COUNTER (IC) AND BEGIN
                        OPERATION OF THE PROGRAM
?36 -|
?-19-|----------------- DATA ENTERED WHEN READ STATEMENTS WERE EXECUTED
 55 ------------------- RESULT PRINTED BY THE PROGRAM
?91
?54
 37
?27
?18
 9
?18
?27
 9
?
←ESC:        9200
```

6. SUGGESTED PROJECTS

6.1 Write and execute a MICRCOMPUTER program which will read two numbers, compute their sum, and print the result.

6.2 Write and run a program which will compute the function $y = 2x - 4$. The program should read x, compute and print y, return to read another value of x, etc.

6.3 Do the same as in 6.2 for the function $y = x^3$.

<u>HINT</u>: It should only be necessary to execute one STORE instruction.

6.4 Write and run a program which will read a pair of numbers a, b, compute $y = (a-b)/(a+b)$, print the result y, return to read another pair of numbers, etc.

<u>NOTE</u>:

1. Before the division can be carried out, both the numerator and the denominator must be computed. At least one of these must be saved in the memory while the other is being computed.
2. Recall that the remainder is lost when a division is executed. For example, you will get $y = 0$ for $a = 2$, $b = 1$. Thus, a MICROCOMPUTER program can compute the <u>quotient</u>, but not the <u>remainder</u> of the division.

SOLUTIONS TO SUGGESTED PROJECTS

Problem 6.1

```
>RUN

MICROCOMPUTER VERSION 5 LEVEL 1 OCT 19 14:24
*0,730
*1,130
*2,730
*3,230
*4,430
*5,830
*100,0

?7
?2
 9
*102,0
GOODBYE, COME AGAIN SOON...
```

Problem 6.2

```
>RUN
MICROCOMPUTER VERSION 5 LEVEL 1 OCT 19 14:26
*0,750
*1,150
*2,099
*3,551
*4,352
*5,453
*6,853
*7,900
*51,2
*52,4
*100,0

?10
 16
?20
 36
?
←ESC.         9200
```

Problem 6.3

```
>RUN
MICROCOMPUTER VERSION 5 LEVEL 1 OCT 19 14:27
*0,730
*1,130
*2,530
*3,530
*4,430
*5,830
*6,900
*100,0

?3
 27
?4
 64
?5
 125
?.
←ESC:        9200
```

Problem 6.4

```
>RUN
MICROCOMPUTER VERSION 5 LEVEL 1 OCT 19 14:29
*9,701
*10,702
*11,101
*12,202
*13,403
*14,101
*15,302
*16,603
*17,404
*18,804
*19,909
*100,9

?1
?1
 0
?-2
?3
 -5
?.
←ESC:        9200
```

LISTING OF /MICRO/

```
100 DIM M(100)
200 PRINT "MICROCOMPUTER VERSION 5 LEVEL 1 ":DATE
1200 GO TO 1600
1300 FOR A=1 TO 100
1400 LET M(A)=0
1500 NEXT A
1600 PRINT "*": DEMAND L,I<—— READ LOCATION AND CONTENTS
1700 IF L=INT(L) THEN 2000
1800 PRINT "LOCATIONS MUST BE INTEGERS 0-99"
1900 GO TO 1600
2000 IF L>=0 THEN 2300
2100 PRINT "NO NEGATIVE LOCATIONS PLEASE"
2200 GO TO 1600
2300 IF L<103 THEN 2600
2400 PRINT "THERE ARE LOCATIONS 0-99"
2500 GO TO 1600
2600 IF L=100 THEN 4400<-IF LOCATION IS 100 START EXECUTION
2650 IF L=101 THEN 3250<-IF LOCATION IS 101 PRINT WORD SPECIFIED (SEE 3250)
2660 IF L=102 THEN 3298<-IF LOCATION IS 102 "LOG OUT"
2700 IF I=INT(I) THEN 3000
2800 PRINT "ONLY INTEGERS CAN BE STORED"
2900 GO TO 1600
3000 IF ABS(L)<1000 THEN 3300
3100 PRINT "999 IS THE LARGEST LEGAL INTEGER"
3200 GO TO 1600
3250 PRINT M(I+1)
3260 GO TO 1600
3298 PRINT "GOODBYE, COME AGAIN SOON..."
3299 STOP
3300 LET M(L+1)=I<—IF THERE IS NO ERROR, STORE I IN MEMORY LOCATION L+1
3400 GO TO 1600
4400 PRINT
4500 LET A=0<———— SET ACC = 0
4600 LET C=I-1
4700 LET C=C+1<——— INCREMENT INSTRUCTION COUNTER
4800 IF C=100 THEN 1300
4900 LET I=M(C+1)<————I IS THE CURRENT INSTRUCTION
5000 LET I2=INT(I/100)<—I2 IS THE OP-CODE OF THE INSTRUCTION
5100 LET O=I-(I2*100)<——O IS THE ADDRESS IN THE INSTRUCTION
5200 IF I2=0 THEN 6400 |
5300 IF I2=1 THEN 6800 |
5400 IF I2=2 THEN 7000 |
5500 IF I2=3 THEN 7300 |
5600 IF I2=4 THEN 7600 |— BRANCH TO THE PART OF THE PROGRAM THAT
5700 IF I2=5 THEN 8100 |  EXECUTES THAT INSTRUCTION
5800 IF I2=6 THEN 8400 |
5900 IF I2=7 THEN 9100 |
6000 IF I2=8 THEN 9800 |
6100 IF I2=9 THEN 10000|
6200 PRINT "SYSTEM ERROR AT LOCATION";C
```

```
6300 GO TO 1600
6400 IF A<0 THEN 6600
6500 GO TO 4700                          TRANSFER ON MINUS
6600 LET C=O-1
6700 GO TO 4700
6800 LET A=M(O+1)                        CLEAR AND ADD
6900 GO TO 4700
7000 LET A=A+M(O+1)
7100 GOSUB 10200                         ADD
7200 GO TO 4700
7300 LET A=A-M(O+1)
7400 GOSUB 10200                         SUBTRACT
7500 GO TO 4700
7600 IF ABS(A)<1000 THEN 7900
7700 PRINT "ACC CANNOT BE STORED. MORE THAN 3 DIGITS. LOCATION";C   STORE
7800 GO TO 1300
7900 LET M(O+1)=A
8000 GO TO 4700
8100 LET A=A*M(O+1)
8200 GOSUB 10200                         MULTIPLY
8300 GO TO 4700
8400 IF M(O+1)<>0 THEN 8700
8500 PRINT "DIVIDE BY ZERO AT LOCATION"; C
8600 GO TO 1300                          DIVIDE
8700 LET A=A/M(O+1)
8800 GO SUB 10200
8900 GO TO 4700
9000 PRINT IN FORM "DLB": O+1
9100 INPUT I
9200 LET I=INT(I)
9300 IF ABS(I)<1000 THEN 9600
9400 PRINT "TOO HIGH AN INPUT--RETYPE"   READ
9500 GO TO 9100
9600 LET M(O+1)=I
9700 GO TO 4700
9800 PRINT M(O+1)                        PRINT
9900 GO TO 4700
10000 LET C=O-1                          TRANSFER UNCONDITIONALLY
10100 GO TO 4700
10200 IF ABS(A)<10000 THEN 10500
10300 PRINT "OVERFLOW AT";C              OVERFLOW PROCEDURE
10400 LET A=0
10500 RETURN
10600 END
```